

# On memetic search for the max-mean dispersion problem

Xiangjing Lai and Jin-Kao Hao \*

*LERIA, Université d'Angers, 2 Bd Lavoisier, 49045 Angers, Cedex 01, France*

*Submitted for publication on 19 January 2015*

---

## Abstract

Given a set  $V$  of  $n$  elements and a distance matrix  $[d_{ij}]_{n \times n}$  among elements, the max-mean dispersion problem (MaxMeanDP) consists in selecting a subset  $M$  from  $V$  such that the mean dispersion (or distance) among the selected elements is maximized. Being a useful model to formulate several relevant applications, MaxMeanDP is known to be NP-hard and thus computationally difficult. In this paper, we present a highly effective memetic algorithm for MaxMeanDP which relies on solution recombination and local optimization to find high quality solutions. Computational experiments on the set of 160 benchmark instances with up to 1000 elements commonly used in the literature show that the proposed algorithm improves or matches the published best known results for all instances in a short computing time, with only one exception, while achieving a high success rate of 100%. In particular, we improve 59 previous best results out of the 60 most challenging instances. Results on a set of 40 new large instances with 3000 and 5000 elements are also presented. The key ingredients of the proposed algorithm are investigated to shed light on how they affect the performance of the algorithm.

*Keywords:* Dispersion Problem; Memetic Algorithm; Tabu Search; Heuristics.

---

## 1 Introduction

Given a weighted complete graph  $G = (V, E, D)$ , where  $V$  is the set of  $n$  vertices,  $E$  is the set of  $\frac{n \times (n-1)}{2}$  edges, and  $D$  represents the set of edge weights  $d_{ij}$  ( $i \neq j$ ), the generic equitable dispersion problem consists in selecting a subset  $M$  from  $V$  such that some objective function  $f$  defined on the subgraph

---

\* Corresponding author.

*Email addresses:* laixiangjing@gmail.com (Xiangjing Lai),  
hao@info.univ-angers.fr (Jin-Kao Hao).

induced by  $M$  is optimized [20]. In the related literature, a vertex  $v \in V$  is also called an element, and the edge weight  $d_{ij} \in D$  is called the distance (or diversity) between elements  $i$  and  $j$ .

According to the objective function to be optimized as well as the constraints on the cardinality of subset  $M$ , several specific equitable dispersion problem can be defined. At first, if the cardinality of  $M$  is fixed to a given number  $m$ , the related equitable dispersion problems include the following four classic variants: (1) the max-sum diversity problem, also known as the maximum diversity problem (MDP), which is to maximize the sum of distances among the selected elements [1,2,7,12,15,21,24]; (2) the max-min diversity problem that aims to maximize the minimum distance among the selected elements [5,19,22,23]; (3) the maximum minsum dispersion problem (MaxMinsumDP) that aims to maximize the minimum aggregate dispersion among the selected elements [3,20]; (4) the minimum differential dispersion problem (MinDiffDP) whose goal is to minimize the difference between the maximum and minimum aggregate dispersion among the selected elements to guarantee that each selected element has the approximately same total distance from the other selected elements [3,8,20]. In addition, when the cardinality of subset  $M$  is not fixed, i.e., the size of  $M$  is allowed to vary from 2 to  $n$ , the related equitable dispersion problems include the max-mean dispersion problem (MaxMeanDP) and the weighted MaxMeanDP [4,6,16,20].

In this study, we focus on MaxMeanDP which can be described as follows [20]. Given a set  $V$  of  $n$  elements and a distance matrix  $(d_{ij})_{n \times n}$  where  $d_{ij}$  represents the distance between elements  $i$  and  $j$  and can take a positive or negative value, the max-mean dispersion problem consists in selecting a subset  $M$  ( $|M|$  is not fixed) from  $V$  such that the mean dispersion among the selected elements, i.e.,  $\frac{\sum_{i,j \in M; i < j} d_{ij}}{|M|}$ , is maximized.

MaxMeanDP can be naturally expressed as a quadratic integer program with binary variables  $x_i$  that takes 1 if element  $i$  is selected and 0 otherwise [16,20], i.e.,

$$\text{Maximize } f(s) = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^n d_{ij} x_i x_j}{\sum_{i=1}^n x_i} \quad (1)$$

$$\text{Subject to } \sum_{i=1}^n x_i \geq 2 \quad (2)$$

$$x_i \in \{0, 1\}, i = 1, 2, \dots, n; \quad (3)$$

where the constraint (2) guarantees that at least two elements are selected.

In addition to its theoretical signification as a strongly NP-hard problem [20], MaxMeanDP is notable for its ability to model a variety of real-world applications, such as web pages ranks [14], community mining [25], and others mentioned in [4].

Given the interest of MaxMeanDP, several useful solution approaches have been proposed in the literature to deal with this hard combinatorial optimization problem. First of all, Prokopyev et al. presented a mixed-integer 0-1 linear programming formulation and solved small instances with up to 100 elements with the CPLEX solver [20]. In the same work, they also introduced a GRASP method and made a comparison between the GRASP method and the MILP method to show the superiority of GRASP. Subsequently, Marti and Sandoya proposed a GRASP with the path relinking method (GRASP-PR) [16], and the computational results show that GRASP-PR outperforms the previously reported methods. Very recently, Della et al. reported a hybrid heuristic approach based on a quadratic knapsack formulation [6], and their computational experiment shows that the hybrid approach is superior to the GRASP-PR method. In another very recent work, Carrasco et al. proposed a diversified tabu search algorithm by combining a short-term tabu search procedure and a long-term tabu search procedure [4], and the computational results show that this algorithm clearly dominates the previous GRASP-PR method.

In this paper, we propose the first population-based memetic algorithm for solving MaxMeanDP (called MAMMDP). The proposed algorithm combines a random crossover operator to generate new offspring solutions and a tabu search method to find good local optima.

The performance of our algorithm is assessed on a set of 160 benchmark instances ( $20 \leq n \leq 1000$ ) commonly used in the literature and a set of additional 40 large-sized instances that we generate ( $n = 3000, 5000$ ). For the first set of existing benchmarks, the experimental results show that the proposed algorithm is able to attain, in a short or very short computing time, all current best known results established by any existing algorithms, except for one instance. Furthermore, it can even improve the previous best known result for a number of these instances. The effectiveness of the proposed algorithm is also verified on much larger instances of the second set with 3000 and 5000 elements.

The remaining part of the paper is organized as follows. In Section 2, we describe in detail the general scheme and the ingredients of the proposed algorithm. In Section 3, we present the computational results based on the 200 benchmark instances and compare them with those of the existing state-of-the-art algorithms from the literature. In Section 4, some important ingredients of the proposed algorithm are analyzed and discussed. Finally, we conclude the paper in the last Section.

## 2 Memetic Algorithm for Max-Mean Dispersion Problem

Memetic search is a well-known metaheuristic framework which aims to provide the search with a desirable trade-off between intensification and diversification through the combined use of a crossover operator (to generate new promising solutions) and a local optimization procedure (to locally improve the generated solutions) [17,18].

### 2.1 General Procedure

The proposed memetic algorithm (denoted by MAMMDP) adopts the principles and guidelines of designing effective MA for discrete combinatorial problems [13]. Indeed, the overall performance of a memetic algorithm depends largely on the implementation of these two key components that need to be carefully designed according to the specific problem structure. Additionally, the proposed MAMMDP algorithm uses a population scheme which borrows from the path relinking method [11] to ensure a strong intensification search.

The general procedure of our MAMMDP algorithm is shown in Algorithm 1, where  $s^*$  and  $s^w$  respectively represent the best solution found so far and the worst solution in the population in terms of the objective value, and *PairSet* is the set of solution pairs  $(s^i, s^j)$ , which is initially composed of all the possible solution pairs  $(s^i, s^j)$  in the population and is dynamically updated as the search progresses.

Our MAMMDP algorithm starts with an initial population  $P$  (line 4) which includes  $p$  different solutions, where each of them is randomly generated and then improved by the tabu search procedure. After the initialization of population (Section 2.3), the algorithm enters a while loop (lines 11 to 25) to make a number of generations. At each generation, a solution pair  $(s^i, s^j)$  is randomly selected from *PairSet* and then the crossover operator (line 14) is applied to the selected solution pair  $(s^i, s^j)$  to generate a new solution  $s^o$  (Section 2.5). Subsequently,  $s^o$  is improved by the tabu search procedure (line 15) (Section 2.4). After that, a population updating rule is used to update the population (lines 20 to 24) (Section 2.6). Meanwhile, the *PairSet* is accordingly updated as follows: First, the solution pair  $(s^i, s^j)$  is removed from *PairSet* (line 13); Then, if an offspring solution  $s^o$  replaces the worst solution  $s^w$  in the population, all the solution pairs containing  $s^w$  are removed from *PairSet* and all the solution pairs that can be generated by combining  $s^o$  with other solutions in the population are added into *PairSet* (lines 23 to 24). The while loop ends when *PairSet* becomes empty, then the population is recreated, while preserving the best solution ( $s^*$ ) found so far in the new population (lines 4

to 8), and the above while loop is repeated if the timeout limit is not reached.

It is worth noting that compared with the traditional random selection scheme, the proposed MAMMDP algorithm uses the set *PairSet* to contain the solution pairs of the population for crossover operations. This strategy ensures that every pair of solutions in the population is combined exactly once, favoring an more intensified search.

---

**Algorithm 1** Memetic algorithm for Max-mean Dispersion Problem

---

```

1: Input: The set  $V = \{v_1, v_2, \dots, v_n\}$  of  $n$  elements and the distance matrix  $D = [d_{ij}]_{n \times n}$ , the population size  $p$ , the timeout limit  $t_{out}$ .
2: Output: the best solution  $s^*$  found
3: repeat
4:    $P = \{s^1, \dots, s^p\} \leftarrow \text{Population\_Initialization}(V)$  /* Section 2.3 */
5:   if it is not in the first loop then
6:      $s^w \leftarrow \arg \min \{f(s^i) : i = 1, \dots, p\}$ 
7:      $P \leftarrow P \cup \{s^*\} \setminus \{s^w\}$ 
8:   end if
9:    $s^* \leftarrow \arg \max \{f(s^i) : i = 1, \dots, p\}$  /*  $s^*$  keeps the best solution found */
10:   $\text{PairSet} \leftarrow \{(s^i, s^j) : 1 \leq i < j \leq p\}$ 
11:  while  $\text{PairSet} \neq \emptyset$  and  $\text{time} < t_{out}$  do
12:    Randomly pick a solution pair  $(s^i, s^j) \in \text{PairSet}$ 
13:     $\text{PairSet} \leftarrow \text{PairSet} \setminus \{(s^i, s^j)\}$ 
14:     $s^o \leftarrow \text{CrossoverOperator}(s^i, s^j)$  /* Section 2.5 */
15:     $s^o \leftarrow \text{TabuSearch}(s^o)$  /* Section 2.4 */
16:    if  $f(s^o) > f(s^*)$  then
17:       $s^* \leftarrow s^o$ 
18:    end if
19:     $s^w \leftarrow \arg \min \{f(s^i) : i = 1, \dots, p\}$ 
20:    if  $s^o$  dose not exist in  $P$  and  $f(s^o) > f(s^w)$  then
21:       $P \leftarrow P \cup \{s^o\} \setminus \{s^w\}$ 
22:       $\text{PairSet} \leftarrow \text{PairSet} \setminus \{(s^w, s^k) : s^k \in P\}$ 
23:       $\text{PairSet} \leftarrow \text{PairSet} \cup \{(s^o, s^k) : s^k \in P\}$ 
24:    end if /* Section 2.6 */
25:  end while
26: until  $\text{time} \geq t_{out}$ 

```

---

## 2.2 Search Space and Solution Representation

Given a MaxMeanDP instance denoted by a set  $V$  of  $n$  elements as well as its distance matrix  $D = [d_{ij}]_{n \times n}$ , the search space  $\Omega$  explored by our MAMMDP algorithm is composed of all possible subsets of  $V$ , i.e,  $\Omega = \{M : M \subseteq V\}$ . Formally, a subset  $M$  of  $V$  can be expressed by a  $n$ -dimensional binary vector,  $(x_1, x_2, \dots, x_n)$ , where  $x_i$  takes 1 if element  $i$  belongs to  $M$ , and 0 otherwise.

In other words, the search space  $\Omega$  is composed of all possible  $n$ -dimensional binary vectors, i.e.,

$$\Omega = \{(x_1, x_2, \dots, x_n) : x_i \in \{0, 1\}, 1 \leq i \leq n\}$$

Clearly, the size of the search space  $\Omega$  is bounded by  $O(2^n)$ .

For any candidate solution  $s = (x_1, x_2, \dots, x_n) \in \Omega$ , its quality is given by the objective value ( $f(s)$ , Formula (1)) of the max-mean dispersion problem.

### 2.3 Population Initialization

In our memetic algorithm, the initial population of  $p$  solutions is generated as follows. First, we generate  $p$  random solutions, where each component  $x_i$  ( $i = 1, 2, \dots, n$ ) of a solution  $(x_1, x_2, \dots, x_n)$  is randomly assigned a value from  $\{0, 1\}$  using a uniform probability distribution. Then, the tabu search method (see Section 2.4) is applied to each of the generated solutions to optimize them to a local optimum solution, and the resulting solutions are used to form the initial population.

### 2.4 Local Optimization using Tabu Search

Local optimization is a key component of a memetic algorithm and ensures generally the role of an intensified search to locate high quality local optimum. In this study, we devise a tabu search (TS) method as the local optimization procedure which proves to be highly effective when it is applied alone.

Given a neighborhood structure ( $N(s)$ ) and a starting solution ( $s_0$ ), our tabu search procedure iteratively replaces the incumbent solution  $s$  by a best eligible neighboring solution ( $s'$ ) until the stopping condition is met, i.e., the best solution ( $s_b$ ) is not improved for  $\alpha$  consecutive iterations (called the depth of TS). At each iteration of TS, the performed move is recorded in the tabu list to prevent the reverse move from being performed for the next  $tt$  iterations. Here,  $tt$  is called the tabu tenure and controlled by a special tabu list management strategy. Note that in this tabu search method a move is identified to be eligible if it is not forbidden by the tabu list or it leads to a solution better than the best solution found so far in terms of the objective function value (aspiration criterion).

The general scheme of our TS method is described in Algorithm 2, and the neighborhood structure employed by our TS method and the tabu list management strategy are described in the following subsections.

---

**Algorithm 2** *TabuSearch*( $s_0, N(s), \alpha$ )

---

```
1: Input: Input solution  $s_0$ , neighborhood  $N(s)$ , search depth  $\alpha$ 
2: Output: The best solution  $s_b$  found during the tabu search process
3:  $s \leftarrow s_0$  /*  $s$  is the current solution */
4:  $s_b \leftarrow s$  /*  $s_b$  is the best solution found so far */
5:  $d = 0$  /*  $d$  counts the consecutive iterations where  $s_b$  is not updated */
6: repeat
7:   Choose a best eligible neighboring solution  $s' \in N(s)$ 
8:    $s \leftarrow s'$ 
9:   Update tabu list
10:  if  $f(s) > f(s_b)$  then
11:     $s_b \leftarrow s$ ,
12:     $d = 0$ 
13:  else
14:     $d = d + 1$ 
15:  end if
16: until  $d = \alpha$ 
17: return  $s_b$ 
```

---

#### 2.4.1 Move and Neighborhood

The neighborhood  $N_1$  of our tabu search algorithm is defined by the one-flip move operator which consists of changing the value of a single variable  $x_i$  to its complementary value  $1 - x_i$ . As such, given a solution  $s$ , the one-flip neighborhood  $N_1(s)$  of  $s$  is composed of all possible solutions that can be obtained by applying the one-flip move to  $s$ . The size of the neighborhood  $N_1(s)$  is thus bounded by  $O(n)$ , where  $n$  is the number of components in  $s$ .

#### 2.4.2 Fast Neighborhood Evaluation Technique

In our TS method, we employ a fast neighborhood evaluation technique to examine the neighborhood  $N_1$ . For this purpose, we maintain a  $n$ -dimensional vector  $W = (p_1, p_2, \dots, p_n)$  to effectively calculate the move value (i.e., the change of objective value) of each possible move applicable to the current solution, where the entry  $p_i$  represents the sum of distances between the element  $i$  and the selected elements for the current solution, i.e.,  $p_i = \sum_{j \in M; j \neq i} d_{ij}$ , where  $M$  is the set of selected elements.

Given the current solution  $s$ , if an one-flip move is performed, i.e., a variable  $x_i$  is flipped as  $x_i \leftarrow (1 - x_i)$ , then the move value  $\Delta_i$  can be rapidly computed as follows:

$$\Delta_i = \begin{cases} \frac{-f(s)}{|M|+1} + \frac{p_i}{|M|+1}, & \text{for } x_i = 0; \\ \frac{f(s)}{|M|-1} - \frac{p_i}{|M|-1}, & \text{for } x_i = 1; \end{cases} \quad (4)$$

where  $f(s)$  is the objective value of the current solution  $s$  and  $|M|$  is the number of selected elements in  $s$ . Subsequently, the vector  $W$  is accordingly updated as:

$$p_j = \begin{cases} p_j + d_{ij}, & \text{for } x_i = 0, j \neq i; \\ p_j - d_{ij}, & \text{for } x_i = 1, j \neq i; \\ p_j, & \text{for } j = i; \end{cases} \quad (6)$$

$$p_j = \begin{cases} p_j + d_{ij}, & \text{for } x_i = 0, j \neq i; \\ p_j - d_{ij}, & \text{for } x_i = 1, j \neq i; \\ p_j, & \text{for } j = i; \end{cases} \quad (7)$$

$$p_j = \begin{cases} p_j + d_{ij}, & \text{for } x_i = 0, j \neq i; \\ p_j - d_{ij}, & \text{for } x_i = 1, j \neq i; \\ p_j, & \text{for } j = i; \end{cases} \quad (8)$$

Note that the vector  $W$  is initialized at the beginning of each call of TS with the complexity of  $O(n^2)$ , and is updated in  $O(n)$  after each move.

### 2.4.3 Tabu List Management Strategy

In our TS procedure, we use a tabu list management strategy to dynamically tune the tabu tenure  $tt$ , which is adapted according to a technique proposed in [9] where the tabu tenure is given by a periodic step function. If the current iteration is  $y$ , then the tabu tenure of a move is denoted by  $tt(y)$ .

Precisely, our tabu tenure function is defined, for each period, by a sequence of values  $(a_1, a_2, \dots, a_{q+1})$  and a sequence of interval margins  $(y_1, y_2, \dots, y_{q+1})$  such that for each  $y$  in  $[y_i, y_{i+1} - 1]$ ,  $tt(y) = a_i + rand(2)$ , where  $rand(2)$  denotes a random integer between 0 to 2. Here,  $q$  is fixed to 15,  $(a)_{i=1, \dots, 15} = \frac{T_{max}}{8}(1, 2, 1, 4, 1, 2, 1, 8, 1, 2, 1, 4, 1, 2, 1)$ , where  $T_{max}$  is a parameter which represents the maximum tabu tenure. Finally, the interval margins are defined by  $y_1 = 1$ ,  $y_{i+1} = y_i + 5a_i$  ( $i \leq 15$ ).

Thus, this function varies periodically and for each period, 15 tabu tenures are used dynamically, each being kept for a number of consecutive iterations. In principle, this function helps the tabu search procedure reach a desirable tradeoff between intensification and diversification during its search.

### 2.5 Crossover Operator

In a memetic algorithm, the crossover operator is another essential ingredient whose main goal is to bring the search process to new promising search regions to diversify the search. In this work, we investigate two crossover operators for MaxMeanDP; the first one is the standard uniform crossover (denoted by



---

**Algorithm 3** The Uniform Crossover Operator for MaxMeanDP

---

```
1: Input: Two parent solutions  $s^1 = (x_1^1, x_2^1, \dots, x_n^1)$  and  $s^2 = (x_1^2, x_2^2, \dots, x_n^2)$ .
2: Output: Offspring solution  $s^o = (x_1^o, x_2^o, \dots, x_n^o)$ 
3: for  $i = 1$  to  $n$  do
4:    $r \leftarrow \text{rand}[0, 1)$  /*  $\text{rand}[0, 1)$  denotes a random number between 0 and 1 */
5:   if  $r < 0.5$  then
6:      $x_i^o \leftarrow x_i^1$ 
7:   else
8:      $x_i^o \leftarrow x_i^2$ 
9:   end if
10: end for
11: return  $s^o = (x_1^o, x_2^o, \dots, x_n^o)$ 
```

---

---

**Algorithm 4** The Greedy Crossover Operator for MaxMeanDP

---

```
1: Input: Two parent solutions  $s^1$  and  $s^2$ , and their subsets of selected elements are
   respectively denoted by  $M_1$  and  $M_2$ 
2: Output: Offspring solution  $s^o$  whose subset of selected elements is denoted by  $M_o$ 
3:  $m \leftarrow (|M_1| + |M_2|)/2$  /* Determine approximately the size of the set  $M_o$  */
   /*Generate a partial solution by preserving the common elements of  $M_1$  and  $M_2$  */
4:  $M_o \leftarrow M_1 \cap M_2$ 
5:  $M'_1 \leftarrow M_1 \setminus M_o$ ,  $M'_2 \leftarrow M_2 \setminus M_o$ 
6: while  $|M_o| < m$  do
7:   if  $M'_1 \neq \emptyset$  then
8:      $v_o \leftarrow \text{argmax}\{\Delta_f(v) : v \in M'_1\}$ 
       /*  $\Delta_f(v)$  represents the move value of adding element  $v$  to  $M_o$  */
9:      $M_o \leftarrow M_o \cup \{v_o\}$ ,  $M'_1 \leftarrow M'_1 \setminus \{v_o\}$ 
10:  end if
11:  if  $M'_2 \neq \emptyset$  then
12:     $u_o \leftarrow \text{argmax}\{\Delta_f(u) : u \in M'_2\}$ 
13:     $M_o \leftarrow M_o \cup \{u_o\}$ ,  $M'_2 \leftarrow M'_2 \setminus \{u_o\}$ 
14:  end if
15: end while
16: return  $s^o(\text{i.e., } M_o)$ 
```

---

UC) operator, and the other is a specific greedy crossover (denoted by GC) operator.

UC is very simple and is described in Algorithm 3. Given two parent solutions  $s^1 = (x_1^1, x_2^1, \dots, x_n^1)$  and  $s^2 = (x_1^2, x_2^2, \dots, x_n^2)$ , the value of each component  $x_i^o$  ( $i = 1, 2, \dots, n$ ) of the offspring solution  $s^o$  is randomly chosen from the set  $\{x_i^1, x_i^2\}$  with the same probability of 0.5. In spite of its simplicity, UC has shown to be quite robust and effective in many settings.

The dedicated GC operator is shown in Algorithm 4. Given two parent solutions  $s^1$  and  $s^2$  whose subsets of selected elements are respectively represented by  $M_1$  and  $M_2$ , we first estimate the size of the set  $M_o$  of selected elements

for the offspring solution  $s^o$  such that the Hamming distances of  $s^o$  to  $s^1$  as well as  $s^2$  are approximately the same, i.e.,  $m \leftarrow \frac{|M_1|+|M_2|}{2}$ . Then a partial solution is generated by preserving the common elements of  $M_1$  and  $M_2$ , i.e.,  $M_o \leftarrow M_1 \cap M_2$ . After that, we complete the offspring solution  $s^o$  in a step by step way by choosing in turn a best element from  $M_1 \setminus M_o$  or  $M_2 \setminus M_o$  and adding it to  $M_o$ . More specifically, we first choose an element ( $v_o$ ) that yields the largest move value from  $M_1 \setminus M_o$  and add it to  $M_o$ , then we choose another element ( $u_o$ ) that yields the largest move value from  $M_2 \setminus M_o$  and add it to  $M_o$ . We repeat these two steps until the size of  $M_o$  reaches  $m$ .

Intuitively, UC is more disruptive than GC and thus is suitable for the purpose of diversifying the search. On the other hand, GC is more conservative and computationally more expensive. Based on the computational outcomes presented in Section 4.2, we adopt in this study the UC operator as the main crossover operator of our memetic algorithm, while using GC as a reference operator for our analysis on crossovers.

## 2.6 Population Updating Rule

When a new offspring solution is generated by the crossover operator, it is first improved the tabu search procedure and then used to update the population according to the following rule. If the offspring solution is distinct from any existing solution in the population and is better than the worst solution in the population in terms of objective value, then the offspring solution replaces the worst solution of the population. Otherwise, the population is kept unchanged.

# 3 Experimental Results and Comparisons

In this section, we run extensive computational experiments to assess the performance of our memetic algorithm based on a large number of MaxMeanDP benchmark instances.

## 3.1 Benchmark Instances

Our computational experiments are carried out on two types of instances, namely Type I and Type II. The distances of Type I instances are randomly generated in the interval  $[-10, 10]$  with a uniform probability distribution, while the distances of Type II instances are generated from  $[-10, -5] \cup [5, 10]$  with the same probability distribution.

Additionally, the set of benchmark instances used in our experiments is composed of two subsets. The first subset consists of 80 Type I instances and 80 Type II instances with the number of elements  $n$  ranging from 20 to 1000. These 160 instances were extensively adopted by the previous studies [4,6,16] and are available online at <http://www.optsim.es>. The second subset consists of 20 Type I and 20 Type II large instances with  $n = 3000$  or  $5000$ . The source code of the generator used to obtain these 40 large instances will be available<sup>1</sup>.

### 3.2 Parameter Settings and Experimental Protocol

Our memetic algorithm relies on only three parameters: the population size  $p$ , the depth of tabu search  $\alpha$  and the maximum tabu tenure  $T_{max}$ . For  $p$  and  $\alpha$ , we follow [24] and set  $p = 10$ ,  $\alpha = 50000$  while setting  $T_{max} = 120$  empirically. This parameter setting is used for all the experiments reported in the paper. Even if fine-tuning these parameters would lead to better results, as we show below, our algorithm with this fixed setting is able to attain a high performance with respect to the state of the art results.

Our memetic algorithm is programmed in C++ and compiled using g++ compiler with the '-O2' flag<sup>2</sup>. All experiments are carried out on a computer with an Intel Xeon E5440 processor (2.83 GHz CPU and 2Gb RAM), running the Linux operating system. Following the DIMACS machine benchmark procedure<sup>3</sup>, our machine requires respectively 0.23, 1.42, and 5.42 seconds for the graphs r300.5, r400.5, r500.5.

Given the stochastic nature of our algorithm, we solve each tested problem instance 20 times, where the stopping condition is given by a cutoff time limit which depends on the size of the instances. Specifically, the cutoff limit  $t_{out}$  is set to be 10 seconds for  $n \leq 150$ , 100 seconds for  $n \in [500, 1000]$ , 1000 seconds for  $n = 3000$ , and 2000 seconds for  $n = 5000$ . As we discuss in Section 3.3, these time limits are significantly shorter than those used by the reference algorithms of the literature.

Table 1  
Computational results of the proposed MAMMDP algorithm on the set of 60 representative instances with  $500 \leq n \leq 1000$ . Each instance is independently solved 20 times, and improved results are indicated in bold compared to the previous best known results  $f_{pre}$  of the literature reported in [4,6,16].

Instance	n	$f_{pre}$	Memetic Algorithm			
		[4,6,16]	$f_{best}$	$f_{avg}$	SR	$t(s)$
MDPI1_500	500	81.28	81.277044	81.277044	20/20	0.69
MDPI2_500	500	77.60	<b>78.610216</b>	<b>78.610216</b>	20/20	1.43
MDPI3_500	500	75.65	<b>76.300787</b>	<b>76.300787</b>	20/20	2.71
MDPI4_500	500	82.28	<b>82.332081</b>	<b>82.332081</b>	20/20	0.95
MDPI5_500	500	80.01	<b>80.354029</b>	<b>80.354029</b>	20/20	2.80
MDPI6_500	500	81.12	<b>81.248553</b>	<b>81.248553</b>	20/20	0.78
MDPI7_500	500	78.09	<b>78.164511</b>	<b>78.164511</b>	20/20	0.92
MDPI8_500	500	79.01	<b>79.139881</b>	<b>79.139881</b>	20/20	1.27
MDPI9_500	500	77.15	<b>77.421000</b>	<b>77.421000</b>	20/20	2.37
MDPI10_500	500	81.24	<b>81.309871</b>	<b>81.309871</b>	20/20	0.91
MDPI11_500	500	109.33	<b>109.610136</b>	<b>109.610136</b>	20/20	0.75
MDPI12_500	500	105.06	<b>105.717536</b>	<b>105.717536</b>	20/20	0.88
MDPI13_500	500	107.64	<b>107.821739</b>	<b>107.821739</b>	20/20	0.89
MDPI14_500	500	105.69	<b>106.100071</b>	<b>106.100071</b>	20/20	0.56
MDPI15_500	500	106.59	<b>106.857162</b>	<b>106.857162</b>	20/20	0.99
MDPI16_500	500	106.17	<b>106.297958</b>	<b>106.297958</b>	20/20	0.98
MDPI17_500	500	106.92	<b>107.149379</b>	<b>107.149379</b>	20/20	0.88
MDPI18_500	500	103.49	<b>103.779195</b>	<b>103.779195</b>	20/20	0.59
MDPI19_500	500	106.20	<b>106.619793</b>	<b>106.619793</b>	20/20	1.11
MDPI10_750	750	103.79	<b>104.651507</b>	<b>104.651507</b>	20/20	1.01
MDPI1_750	750	95.86	<b>96.644236</b>	<b>96.644236</b>	20/20	7.98
MDPI2_750	750	97.42	<b>97.564880</b>	<b>97.564880</b>	20/20	3.82
MDPI3_750	750	96.97	<b>97.798864</b>	<b>97.798864</b>	20/20	1.81
MDPI4_750	750	95.21	<b>96.041364</b>	<b>96.041364</b>	20/20	4.38
MDPI5_750	750	96.65	<b>96.740448</b>	<b>96.740448</b>	20/20	1.21
MDPI6_750	750	99.25	<b>99.861250</b>	<b>99.861250</b>	20/20	5.55
MDPI7_750	750	96.26	<b>96.545413</b>	<b>96.545413</b>	20/20	1.01
MDPI8_750	750	96.46	<b>96.726976</b>	<b>96.726976</b>	20/20	1.73
MDPI9_750	750	96.78	<b>98.058377</b>	<b>98.058377</b>	20/20	2.18
MDPI10_750	750	99.85	<b>100.064185</b>	<b>100.064185</b>	20/20	3.42
MDPI11_750	750	127.69	<b>128.863707</b>	<b>128.863707</b>	20/20	5.66
MDPI12_750	750	130.79	<b>130.954426</b>	<b>130.954426</b>	20/20	2.31
MDPI13_750	750	129.40	<b>129.782453</b>	<b>129.782453</b>	20/20	11.64
MDPI14_750	750	125.68	<b>126.582271</b>	<b>126.582271</b>	20/20	1.48
MDPI15_750	750	128.13	<b>129.122878</b>	<b>129.122878</b>	20/20	1.32
MDPI16_750	750	128.55	<b>129.025215</b>	<b>129.025215</b>	20/20	7.98
MDPI17_750	750	124.91	<b>125.646682</b>	<b>125.646682</b>	20/20	3.38
MDPI18_750	750	130.66	<b>130.940548</b>	<b>130.940548</b>	20/20	1.91
MDPI19_750	750	128.89	<b>128.889908</b>	<b>128.889908</b>	20/20	1.30
MDPI10_1000	1000	132.99	<b>133.265300</b>	<b>133.265300</b>	20/20	1.81
MDPI1_1000	1000	118.76	<b>119.174112</b>	<b>119.174112</b>	20/20	8.25
MDPI2_1000	1000	113.22	<b>113.524795</b>	<b>113.524795</b>	20/20	3.52
MDPI3_1000	1000	114.51	<b>115.138638</b>	<b>115.138638</b>	20/20	2.32
MDPI4_1000	1000	110.53	<b>111.127437</b>	<b>111.127437</b>	20/20	5.72
MDPI5_1000	1000	111.24	<b>112.723188</b>	<b>112.723188</b>	20/20	1.61
MDPI6_1000	1000	112.08	<b>113.198718</b>	<b>113.198718</b>	20/20	7.72
MDPI7_1000	1000	110.94	<b>111.555536</b>	<b>111.555536</b>	20/20	1.88
MDPI8_1000	1000	110.29	<b>111.263194</b>	<b>111.263194</b>	20/20	3.55
MDPI9_1000	1000	115.78	<b>115.958833</b>	<b>115.958833</b>	20/20	2.38
MDPI10_1000	1000	114.29	<b>114.731644</b>	<b>114.731644</b>	20/20	2.16
MDPI11_1000	1000	145.46	<b>147.936175</b>	<b>147.936175</b>	20/20	1.60
MDPI12_1000	1000	150.49	<b>151.380035</b>	<b>151.380035</b>	20/20	1.78
MDPI13_1000	1000	149.36	<b>150.788178</b>	<b>150.788178</b>	20/20	4.92
MDPI14_1000	1000	147.91	<b>149.178006</b>	<b>149.178006</b>	20/20	3.80
MDPI15_1000	1000	150.23	<b>151.520847</b>	<b>151.520847</b>	20/20	3.28
MDPI16_1000	1000	147.29	<b>148.343378</b>	<b>148.343378</b>	20/20	3.22
MDPI17_1000	1000	148.41	<b>148.742375</b>	<b>148.742375</b>	20/20	6.30
MDPI18_1000	1000	145.87	<b>147.826804</b>	<b>147.826804</b>	20/20	13.52
MDPI19_1000	1000	145.67	<b>147.078145</b>	<b>147.078145</b>	20/20	1.63
MDPI10_1000	1000	148.40	<b>150.046137</b>	<b>150.046137</b>	20/20	2.13
#Better			59	59		
#Equal			1	1		
#Worse			0	0		

### 3.3 Computational Results and Comparisons on Small and Medium Sized Instances

Our first experiment aims to evaluate the performance of our MAMMDP algorithm on the set of 160 popular instances with up to 1000 elements. The

<sup>1</sup> The source code of generating these instances will be available from our website.

<sup>2</sup> Our best results and the source code of our algorithm will be made available online.

<sup>3</sup> dmclique, <ftp://dimacs.rutgers.edu/pub/dsj/clique>, the benchmark procedure is compiled by gcc compiler with the '-O2' flag

computational results of MAMMDP on the 60 medium sized instances are summarized in Table 1, whereas the results of the 100 small instances with  $n \leq 150$  are given in the Appendix 6 with the same computational statistics.

The first two columns of the table give respectively the name and size of instances. Column 3 indicates the best objective values ( $f_{pre}$ ) of the literature which are compiled from the best results yielded by three recent and best performing algorithms, namely GRASP-PR [16], a hybrid heuristic approach [6], and a diversified tabu search method [4] (from <http://www.optsi.com.es>). Note that the previous best known results ( $f_{pre}$ ) are given with two decimal in the literature. In [4,16], the cutoff time limits are set to 90, 600, and 1800 seconds for instances with the size 500, 750, and 1000, respectively, and in [6] the cutoff time limits are respectively set to 60 and 600 seconds for the instances with the size 150 and 500. The GRASP-PR method was performed on a computer with an Intel Core Solo 1.4 GHz CPU with 3 GB RAM [16]; the hybrid heuristic approach was run on a computer with an Intel Core i5-3550 3.30GHz CPU with 4GB RAM [6] and the diversified tabu search method was run on a computer with an Intel Core 2 Quad CPU and 6 GB RAM [4].

Our results are reported in columns 4 to 7, including the best objective value ( $f_{best}$ ) yielded over 20 independent runs, the average objective value ( $f_{avg}$ ), the success rate ( $SR$ ) to achieve  $f_{best}$ , and the average computing time in seconds ( $t(s)$ ) to achieve  $f_{best}$ . The last three rows *Better*, *Equal*, *Worse* of the table respectively show the number of instances for which our result is better, equal to and worse than  $f_{pre}$ . The improved results are indicated in bold compared to  $f_{pre}$ .

First, one observes from Table 1 that our MAMMDP algorithm improves the previous best known result for all instances except for one instance for which our result matches the previous best known result. Therefore, these results clearly indicate the superiority of the proposed MAMMDP algorithm compared to the previous MaxMeanDP algorithms. Second, when examining the success rate of the algorithm, one can find that the MAMMDP algorithm achieves a success rate of 100% for all tested instances, which means a good robustness of our MAMMDP algorithm. Third, in terms of average computing time, one observes that for all instances our MAMMDP algorithm obtains its best result with an average time of less than 14 seconds, which are much shorter than those of the previous algorithms in the literature.

### 3.4 Computational Results on Large-Scale Instances

In order to further assess the performance of the proposed MAMMDP algorithm on the large-scale instances, we run independently MAMMDP 20 times

Table 2

Computational results of the proposed memetic algorithm on the set of 40 large instances with  $n = 3000, 5000$ . Each instance is independently solved 20 times.

Instance	n	Memetic Algorithm			
		$f_{best}$	$f_{avg}$	SR	$t(s)$
MDPI1_3000	3000	189.048965	189.048965	20/20	88.36
MDPI2_3000	3000	187.387292	187.387292	20/20	60.71
MDPI3_3000	3000	185.666806	185.655084	13/20	352.85
MDPI4_3000	3000	186.163727	186.153631	16/20	300.37
MDPI5_3000	3000	187.545515	187.545515	20/20	61.29
MDPI6_3000	3000	189.431257	189.431257	20/20	51.99
MDPI7_3000	3000	188.242583	188.242583	20/20	86.57
MDPI8_3000	3000	186.796814	186.796814	20/20	48.04
MDPI9_3000	3000	188.231264	188.231264	20/20	151.78
MDPI10_3000	3000	185.682511	185.623778	10/20	228.72
MDPI11_3000	3000	252.320433	252.320433	20/20	59.70
MDPI12_3000	3000	250.062137	250.062137	20/20	220.10
MDPI13_3000	3000	251.906270	251.906270	20/20	146.32
MDPI14_3000	3000	253.941007	253.940596	19/20	370.76
MDPI15_3000	3000	253.260423	253.260350	17/20	374.00
MDPI16_3000	3000	250.677750	250.677750	20/20	55.35
MDPI17_3000	3000	251.134413	251.134413	20/20	74.72
MDPI18_3000	3000	252.999648	252.999648	20/20	79.82
MDPI19_3000	3000	252.425770	252.425770	20/20	90.27
MDPI10_3000	3000	252.396590	252.396590	20/20	13.18
MDPI1_5000	5000	240.162535	240.102875	7/20	312.13
MDPI2_5000	5000	241.827401	241.792978	6/20	1244.36
MDPI3_5000	5000	240.890819	240.888162	19/20	810.48
MDPI4_5000	5000	240.997186	240.976789	6/20	653.64
MDPI5_5000	5000	242.480129	242.475885	19/20	735.16
MDPI6_5000	5000	240.322850	240.306326	8/20	976.02
MDPI7_5000	5000	242.814943	242.774982	5/20	259.50
MDPI8_5000	5000	241.194990	241.161763	8/20	1148.60
MDPI9_5000	5000	239.760560	239.667613	4/20	1219.71
MDPI10_5000	5000	243.473734	243.373015	4/20	457.28
MDPI11_5000	5000	322.235897	322.181291	5/20	1519.05
MDPI12_5000	5000	327.301910	327.006342	5/20	1103.13
MDPI13_5000	5000	324.813456	324.801590	10/20	955.81
MDPI14_5000	5000	322.237586	322.197276	5/20	664.10
MDPI15_5000	5000	322.491211	322.380726	7/20	1014.90
MDPI16_5000	5000	322.950488	322.703887	4/20	352.88
MDPI17_5000	5000	322.850438	322.793125	10/20	714.31
MDPI18_5000	5000	323.112120	323.053268	11/20	879.48
MDPI19_5000	5000	323.543775	323.339842	7/20	569.73
MDPI10_5000	5000	324.519908	324.414458	15/20	752.95

to solve each instance of the second set of benchmarks with  $n \geq 3000$ , and report the computational statistics in Table 2 with the same information as in Table 1.

Table 2 discloses that for the instances with 3000 elements, our MAMMDP algorithm reaches a success rate of at least 10/20, which is an interesting indicator as to its good performance for these instances. However, for the still larger instances with  $n = 5000$ , the success rate of the algorithm significantly varies between 4/20 and 19/20, which means that these large instances are clearly more difficult. Nevertheless, one observes that the difference between the best and average objective values is very small for all instances. These results can be served as reference lower bounds for future comparisons of new MaxMeanDP algorithms.

## 4 Analysis and Discussions

In this section, we study some essential ingredients of the proposed algorithm to understand their impacts on the performance of the proposed algorithm.

#### 4.1 Effectiveness of the Tabu Search Procedure

Table 3

Computational results of the underlying tabu search procedure of the proposed algorithm on the set of 30 representative instances with  $500 \leq n \leq 1000$ . Each instance is independently solved 20 times, and improved results are indicated in bold compared to the previous best known objective values.

Instance	n	$f_{pre}$	$f^*$	Tabu Search		SR	$t(s)$
				$f_{best}$	$f_{avg}$		
MDPI1_500	500	81.28	81.277044	81.277044	81.277044	20/20	0.54
MDPI2_500	500	77.60	78.610216	<b>78.610216</b>	<b>78.586017</b>	8/20	0.68
MDPI3_500	500	75.65	76.300787	<b>76.300787</b>	<b>76.252735</b>	7/20	0.80
MDPI4_500	500	82.28	82.332081	<b>82.332081</b>	<b>82.325976</b>	15/20	0.65
MDPI5_500	500	80.01	80.354029	<b>80.354029</b>	<b>80.349285</b>	5/20	0.78
MDPI11_500	500	109.33	109.610136	<b>109.610136</b>	<b>109.473283</b>	15/20	0.56
MDPI2_500	500	105.06	105.717536	<b>105.717536</b>	<b>105.678373</b>	14/20	0.70
MDPI3_500	500	107.64	107.821739	<b>107.821739</b>	<b>107.808748</b>	18/20	0.77
MDPI4_500	500	105.69	106.100071	<b>106.100071</b>	<b>106.08738</b>	18/20	0.54
MDPI5_500	500	106.59	106.857162	<b>106.857162</b>	<b>106.834002</b>	14/20	0.64
MDPI1_750	750	95.86	96.644236	<b>96.644236</b>	<b>96.472938</b>	5/20	1.24
MDPI2_750	750	97.42	97.564880	<b>97.564880</b>	<b>97.557159</b>	9/20	1.10
MDPI3_750	750	96.97	97.798864	<b>97.798864</b>	<b>97.780265</b>	15/20	1.05
MDPI4_750	750	95.21	96.041364	<b>96.041364</b>	<b>95.880611</b>	2/20	1.17
MDPI5_750	750	96.65	96.740448	<b>96.740448</b>	<b>96.731085</b>	18/20	0.99
MDPI11_750	750	127.69	128.863707	<b>128.863707</b>	<b>128.401188</b>	4/20	1.12
MDPI2_750	750	130.79	130.954426	<b>130.954426</b>	<b>130.920628</b>	7/20	1.01
MDPI3_750	750	129.40	129.782453	<b>129.782453</b>	<b>129.557185</b>	1/20	1.03
MDPI4_750	750	125.68	126.582271	<b>126.582271</b>	<b>126.441529</b>	11/20	0.96
MDPI5_750	750	128.13	129.122878	<b>129.122878</b>	<b>129.032812</b>	15/20	1.11
MDPI1_1000	1000	118.76	119.174112	<b>119.174112</b>	<b>118.836369</b>	5/20	1.63
MDPI2_1000	1000	113.22	113.524795	<b>113.524795</b>	<b>113.363683</b>	7/20	1.74
MDPI3_1000	1000	114.51	115.138638	<b>115.138638</b>	<b>115.106057</b>	16/20	1.33
MDPI4_1000	1000	110.53	111.127437	<b>111.127437</b>	<b>110.924380</b>	4/20	1.42
MDPI5_1000	1000	111.24	112.723188	<b>112.723188</b>	<b>112.723188</b>	20/20	1.18
MDPI11_1000	1000	145.46	147.936175	<b>147.936175</b>	<b>147.919913</b>	17/20	1.44
MDPI2_1000	1000	150.49	151.380035	<b>151.380035</b>	<b>151.344939</b>	18/20	1.54
MDPI3_1000	1000	149.36	150.788178	<b>150.788178</b>	<b>150.475002</b>	7/20	1.82
MDPI4_1000	1000	147.91	149.178006	<b>149.178006</b>	<b>149.093037</b>	11/20	1.59
MDPI5_1000	1000	150.23	151.520847	<b>151.520847</b>	<b>151.495094</b>	10/20	1.57
#Better				29	29		
#Equal				1	1		
#Worse				0	0		

First, to show the effectiveness of the underlying tabu search procedure of the proposed algorithm, we carry out an additional experiment on 30 representative instances, where each instance is independently solved 20 times by the tabu search procedure with randomly generated initial solutions. The computational results are summarized in Table 3, where  $f^*$  and  $t(s)$  represent respectively the previous best known result and the average computing time over 20 runs, other symbols are the same as those in Table 1. Note that the improved results compared to the previous best known values  $f_{pre}$  are indicated in bold.

Table 3 discloses that our tabu search procedure alone is able to attain the previous best known result for the tested instances. Moreover, for all tested instances, the obtained average objective value ( $f_{avg}$ ) is still better than the previous best known result ( $f_{pre}$ ) on 29 out of 30 instances. These results indicate that our tabu search method is quite effective with respect to the existing algorithms designed for MaxMeanDP. In terms of computational time, one observes that one run of the tabu search procedure takes on average less than 2 seconds for instances with  $n \leq 1000$ , much shorter than those required



by the state of the art algorithms [4,6,16]. To sum, this experiment shows that our tabu search procedure is highly effective compared with the existing state-of-the-art algorithms in the literature.

It should be mentioned that compared to the previous methods for MaxMe-anDP, such as those in [4], the success of our tabu search method may be attributed to the combined use of the neighborhood  $N_1$ , the fast neighborhood evaluation technique and its dynamic tabu list management strategy.

## 4.2 Influence of the Crossover Operator

Table 4

Comparison between the UC and GC crossover operators on the set of 40 large instances with  $n = 3000$  or  $5000$ . Each instance is independently solved 20 times by the memetic algorithms with the UC and GC operators respectively, and better results between these two memetic algorithms are indicated in bold.

Instance	Uniform Crossover				Greedy Crossover			
	$f_{best}$	$f_{avg}$	SR	$t(s)$	$f_{best}$	$f_{avg}$	SR	$t(s)$
MDPI1_3000	189.048965	189.048965	20/20	88.36	189.048965	189.048965	20/20	82.49
MDPI2_3000	187.387292	187.387292	20/20	60.71	187.387292	187.387292	20/20	71.95
MDPI3_3000	185.666806	185.655084	13/20	352.85	185.666806	<b>185.656214</b>	14/20	430.55
MDPI4_3000	186.163727	186.153631	16/20	300.37	186.163727	<b>186.163727</b>	20/20	248.47
MDPI5_3000	187.545515	187.545515	20/20	61.29	187.545515	187.545515	20/20	87.26
MDPI6_3000	189.431257	189.431257	20/20	51.99	189.431257	189.431257	20/20	51.15
MDPI7_3000	188.242583	188.242583	20/20	86.57	188.242583	188.242583	20/20	103.65
MDPI8_3000	186.796814	186.796814	20/20	48.04	186.796814	186.796814	20/20	68.57
MDPI9_3000	188.231264	188.231264	20/20	151.78	188.231264	188.231264	20/20	69.86
MDPI10_3000	185.682511	<b>185.623778</b>	10/20	228.72	185.682511	185.623041	14/20	386.07
MDPI11_3000	252.320433	252.320433	20/20	59.70	252.320433	252.320433	20/20	78.92
MDPI12_3000	250.062137	250.062137	20/20	220.10	250.062137	250.062137	20/20	219.11
MDPI13_3000	251.906270	251.906270	20/20	146.32	251.906270	251.906270	20/20	139.13
MDPI14_3000	253.941007	<b>253.940596</b>	19/20	370.76	253.941007	253.938635	19/20	243.89
MDPI15_3000	253.260423	253.260350	17/20	374.00	253.260423	<b>253.260423</b>	20/20	301.14
MDPI16_3000	250.677750	250.677750	20/20	55.35	250.677750	250.677750	20/20	63.01
MDPI17_3000	251.134413	251.134413	20/20	74.72	251.134413	251.134413	20/20	75.89
MDPI18_3000	252.999648	252.999648	20/20	79.82	252.999648	252.999648	20/20	62.52
MDPI19_3000	252.425770	252.425770	20/20	90.27	252.425770	252.425770	20/20	96.89
MDPI10_5000	252.396590	252.396590	20/20	13.18	252.396590	252.396590	20/20	18.34
MDPI1_5000	240.162535	240.162535	7/20	312.13	240.162535	<b>240.110086</b>	9/20	104.32
MDPI2_5000	241.827401	<b>241.792978</b>	6/20	1244.36	241.827401	241.750738	3/20	873.37
MDPI3_5000	240.890819	<b>240.888162</b>	19/20	810.48	240.890819	240.875111	14/20	681.51
MDPI4_5000	240.997186	240.976789	6/20	653.64	240.997186	<b>240.987885</b>	10/20	1062.58
MDPI5_5000	242.480129	<b>242.475885</b>	19/20	735.16	242.480129	242.475876	18/20	790.37
MDPI6_5000	240.322850	<b>240.306326</b>	8/20	976.02	<b>240.376038</b>	240.296124	4/20	564.79
MDPI7_5000	242.814943	242.774982	5/20	259.50	<b>242.820139</b>	<b>242.782093</b>	5/20	261.98
MDPI8_5000	241.194990	<b>241.161763</b>	8/20	1148.60	241.194990	241.159647	10/20	1219.58
MDPI9_5000	239.760560	<b>239.667613</b>	4/20	1219.71	239.760560	239.569628	4/20	756.59
MDPI10_5000	<b>243.473734</b>	<b>243.373015</b>	4/20	457.28	243.385487	243.356904	9/20	1057.73
MDPI11_5000	322.235897	<b>322.181291</b>	5/20	1519.05	322.235897	322.172704	5/20	541.47
MDPI12_5000	327.301910	<b>327.006342</b>	5/20	1103.13	327.301910	326.953531	7/20	662.58
MDPI13_5000	324.813456	<b>324.801590</b>	10/20	955.81	324.813456	324.790747	4/20	629.53
MDPI14_5000	322.237586	322.197276	5/20	664.10	322.237586	<b>322.205163</b>	7/20	535.54
MDPI15_5000	322.491211	322.380726	7/20	1014.90	322.491211	<b>322.403539</b>	8/20	1095.86
MDPI16_5000	322.950488	<b>322.703887</b>	4/20	352.88	322.950488	322.692191	6/20	457.37
MDPI17_5000	322.850438	<b>322.793125</b>	10/20	714.31	322.850438	322.774304	12/20	631.95
MDPI18_5000	323.112120	<b>323.053268</b>	11/20	879.48	323.112120	323.007841	13/20	764.65
MDPI19_5000	323.543775	<b>323.339842</b>	7/20	569.73	323.543775	323.273630	5/20	305.89
MDPI10_5000	324.519908	324.414458	15/20	752.95	324.519908	<b>324.500667</b>	19/20	659.21
#Better	1	16			2	9		
#Equal	37	15			37	15		
#Worse	2	9			1	16		
p-value	5.637e-1	1.615e-1						

In this section, we show a study about the influence of the crossover operator on the performance of the proposed algorithm by comparing the uniform crossover (UC) operator and the greedy crossover (GC) operator. The experiment is carried out on a set of 40 large instance with  $n \geq 3000$ , where the memetic algorithms with UC and GC are respectively performed 20 times



for each of the tested instances. The computational results of the both algorithms are summarized in Table 4, including the best ( $f_{best}$ ) and average ( $f_{avg}$ ) objective values obtained over 20 runs, the success rate ( $SR$ ) and average computing time in seconds ( $t(s)$ ) to achieve the associated  $f_{best}$ , where better results between these two algorithms are indicated in bold. The rows, *Better*, *Equal*, *Worse* denote respectively the number of instances for which an algorithm yields better, equal, worse results compared to the other algorithm. Finally, to verify whether there exists a significant difference between the two crossover operators in terms of the best and average objective values, the *p-values* from the non-parametric Friedman test are reported in the last row.

One observes from Table 4 that the UC and GC operators are comparable in the overall performance of the algorithm in terms of best and average results, which is confirmed by the Friedman test (*p-values*  $> 0.05$ ). First, in terms of the best objective value, UC and GC yield respectively better result on 1 and 2 instances compared to another operator. Second, in terms of the average objective value, it can be found the UC operator produces a better and worse result respectively for 16 and 9 instances. As to the success rate and computing time, these two crossover operators also achieve similar performances. This experiment demonstrates that there is no dominance of one crossover over the other. Instead, they are complementary to solve different instances. One interesting future study would be to investigate the ways of using jointly these two operators with the search algorithm.

### 4.3 Improvement of Memetic Algorithm Over the Tabu Search Procedure

As shown in Section 4.1, our tabu search procedure is very competitive compared to the existing algorithms in the literature. So it is interesting to know whether our MAMMDP algorithm has a significant improvement over its underlying local optimization (the tabu search) procedure. To compare the performances of the MAMMDP algorithm and its underlying tabu search procedure, the multi-start tabu search (MTS) and MAMMDP algorithms are respectively performed 20 times for each of the 40 representative instances with  $n = 3000$  or  $5000$  under the same cutoff time limits given in Section 3.2. Notice that for MTS, the tabu search procedure is run in a multi-start way with a randomly generated initial solution for each re-start until the timeout limit is reached, the tabu search procedure being re-started once the depth of tabu search  $\alpha$  (which is set to  $5 \times 10^4$ ) is reached. The computational results of both algorithms are respectively summarized in Table 5 which is composed of two parts, where the symbols are the same with those in Table 4.

Table 5 discloses that for the 20 instances with  $n = 3000$  the MAMMDP

Table 5

Comparison between the multi-start tabu search method (MTS) and the proposed memetic algorithm on the set of 40 large instances with  $n \geq 3000$ . Each instance is independently solved 20 times by both algorithms respectively, and better results between two algorithms are indicated in bold.

Instance	Memetic Algorithm				MTS			
	$f_{best}$	$f_{avg}$	SR	$t(s)$	$f_{best}$	$f_{avg}$	SR	$t(s)$
MDPI1_3000	189.048965	189.048965	20/20	88.36	189.048965	189.048965	20/20	120.05
MDPI2_3000	187.387292	187.387292	20/20	60.71	187.387292	187.387292	20/20	101.07
MDPI3_3000	185.666806	<b>185.655084</b>	13/20	352.85	185.666806	185.651588	10/20	526.05
MDPI4_3000	186.163727	186.153631	16/20	300.37	186.163727	<b>186.163727</b>	20/20	136.64
MDPI5_3000	187.545515	187.545515	20/20	61.29	187.545515	187.545515	20/20	133.70
MDPI6_3000	189.431257	189.431257	20/20	51.99	189.431257	189.431257	20/20	35.59
MDPI7_3000	188.242583	188.242583	20/20	86.57	188.242583	188.242583	20/20	137.56
MDPI8_3000	186.796814	186.796814	20/20	48.04	186.796814	186.796814	20/20	66.76
MDPI9_3000	188.231264	188.231264	20/20	151.78	188.231264	188.231264	20/20	101.11
MDPI10_3000	185.682511	185.623778	10/20	228.72	185.682511	<b>185.672371</b>	18/20	352.70
MDPI11_3000	252.320433	252.320433	20/20	59.70	252.320433	252.320433	20/20	72.49
MDPI12_3000	250.062137	<b>250.062137</b>	20/20	220.10	250.062137	250.054744	7/20	513.80
MDPI13_3000	251.906270	251.906270	20/20	146.32	251.906270	251.906270	20/20	127.32
MDPI14_3000	253.941007	<b>253.940596</b>	19/20	370.76	253.941007	253.939680	18/20	352.64
MDPI15_3000	253.260423	<b>253.260350</b>	17/20	374.00	253.260423	253.260164	14/20	349.19
MDPI16_3000	250.677750	250.677750	20/20	55.35	250.677750	250.677750	20/20	69.78
MDPI17_3000	251.134413	251.134413	20/20	74.72	251.134413	251.134413	20/20	97.74
MDPI18_3000	252.999648	252.999648	20/20	79.82	252.999648	252.999648	20/20	115.84
MDPI19_3000	252.425770	252.425770	20/20	90.27	252.425770	252.425770	20/20	106.79
MDPI10_3000	252.396590	252.396590	20/20	13.18	252.396590	252.396590	20/20	16.06
#Better	0	4	4	14	0	2	2	6
#Equal	20	14	14	0	20	14	14	0
#Worse	0	2	2	6	0	4	4	14
$p\text{-value}$	1.0	4.142e-1						
MDPI1_5000	<b>240.162535</b>	<b>240.102875</b>	7/20	312.13	240.141212	240.021201	2/20	163.67
MDPI2_5000	<b>241.827401</b>	<b>241.792978</b>	6/20	1244.36	241.817543	241.753546	2/20	734.33
MDPI3_5000	240.890819	<b>240.888162</b>	19/20	810.48	240.890819	240.825167	4/20	531.94
MDPI4_5000	<b>240.997186</b>	<b>240.976789</b>	6/20	653.64	240.973489	240.915459	3/20	560.43
MDPI5_5000	242.480129	<b>242.475885</b>	19/20	735.16	242.480129	242.430474	5/20	515.62
MDPI6_5000	240.322850	<b>240.306326</b>	8/20	976.02	<b>240.328684</b>	240.266264	5/20	290.72
MDPI7_5000	242.814943	<b>242.774982</b>	5/20	259.50	<b>242.820139</b>	242.759895	3/20	819.01
MDPI8_5000	<b>241.194990</b>	<b>241.161763</b>	8/20	1148.60	241.144781	241.113453	3/20	721.59
MDPI9_5000	239.760560	<b>239.667613</b>	4/20	1219.71	239.760560	239.514958	4/20	360.05
MDPI10_5000	<b>243.473734</b>	<b>243.373015</b>	4/20	457.28	243.385487	243.348149	9/20	939.59
MDPI11_5000	<b>322.235897</b>	<b>322.181291</b>	5/20	1519.05	322.223220	322.131204	3/20	197.42
MDPI12_5000	327.301910	327.006342	5/20	1103.13	327.301910	<b>327.075247</b>	5/20	18.52
MDPI13_5000	<b>324.813456</b>	<b>324.801590</b>	10/20	955.81	324.810826	324.790223	4/20	27.82
MDPI14_5000	<b>322.237586</b>	<b>322.197276</b>	5/20	664.10	322.212289	322.126605	4/20	338.02
MDPI15_5000	<b>322.491211</b>	<b>322.380726</b>	7/20	1014.90	322.420806	322.301249	5/20	105.12
MDPI16_5000	322.950488	<b>322.703887</b>	4/20	352.88	322.950488	322.615227	5/20	212.55
MDPI17_5000	322.850438	<b>322.793125</b>	10/20	714.31	322.850438	322.778396	8/20	756.14
MDPI18_5000	<b>323.112120</b>	<b>323.053268</b>	11/20	879.48	323.033840	322.873156	5/20	161.94
MDPI19_5000	<b>323.543775</b>	<b>323.339842</b>	7/20	569.73	323.522709	323.278556	3/20	148.94
MDPI10_5000	324.519908	<b>324.414458</b>	15/20	752.95	324.519908	324.294790	10/20	753.14
#Better	11	19			2	1		
#Equal	7	0			7	0		
#Worse	2	1			11	19		
$p\text{-value}$	1.26e - 2	5.699e - 5						

algorithm performs slightly better than the MTS algorithm, but the differences is small. However, for the 20 larger instances with  $n = 5000$ , the MAMMDP algorithm significantly outperforms the the MTS algorithm. First, compared with the MTS algorithm, the MAMMDP algorithm obtains better and worse results in terms of the best objective value on 11 and 2 instances respectively. Second, in terms of average objective value, the MAMMDP algorithm yields better results on 19 out of 20 instances. In addition, from the Friedman test, one observes that the obtained  $p\text{-values}$  are  $1.26e - 2$  ( $<0.05$ ) and  $5.699e - 5$  ( $<0.05$ ) respectively for the best and average objective values, implying there exists a significant difference between these two methods. These outcomes demonstrate that the memetic framework is particularly useful to solve large and difficult instances.

## 5 Conclusions

In this paper, we propose the first population-based memetic algorithm (MAMMDP) for solving the NP-hard max-mean dispersion problem (MaxMeanDP). MAMMDP integrates an effective tabu search procedure and a random crossover operator while adopting an original scheme for parent selection. The computational results on a large number of 200 benchmark instances show that the proposed algorithm is very competitive compared with the state-of-the-art algorithms in the literature. Specifically, it improves or matches the previous best known results for all tested instances with  $n \leq 1000$  with an average computing time of less than 14 seconds and a success rate of 100%, with only one exception. In particular, we found new and improved best results for 59 out of the 60 most challenging instances. We also show computational results on 40 large instances with 3000 or 5000 elements which can serve as reference lower bounds for evaluating new MaxMeanDP algorithms.

The investigations of several important ingredients confirm that both the underlying tabu search procedure and the crossover operator of the proposed algorithm contribute to the high performance of the proposed algorithm. It is shown that the population-based memetic framework is particularly suitable to solve large and difficult problem instances.

The proposed algorithm could be adapted to the weighted version of the max-mean dispersion problem with several small modifications. Some ideas of the proposed algorithm could be applied to other other binary optimization problems (including some dispersion problems) where no constraint is imposed on the number of variables taking the value of one.

## Acknowledgments

The work is partially supported by the LigeRo project (2009-2014) and a post-doc grant for X.J. Lai from the Region of Pays de la Loire (France) and the PGMO (2014-0024H) project from the Jacques Hadamard Mathematical Foundation.

## References

- [1] Aringhieri R., Cordone R., Melzani Y., 2008, Tabu search versus GRASP for the maximum diversity problem. *4OR: A Quarterly Journal of Operations Research* 6(1),45–60.

- [2] Aringhieri R., Cordone R., 2011, Comparing local search metaheuristics for the maximum diversity problem. *Journal of the Operational Research Society* 62, 266–280.
- [3] Aringhieri R., Cordone R., Grosso A., 2015, Construction and improvement algorithms for dispersion problems. *European Journal of Operational Research* 242(1), 21–33.
- [4] Carrasco, R., Anthanh P.T., Gallego M., Gortázar F., Duarte A., Martí R., 2014, Tabu search for the max-mean dispersion problem. <http://www.uv.es/rmarti/paper/docs/mdp10.pdf>.
- [5] Della Croce F., Grosso A., Locatelli M., 2009, A heuristic approach for the max-min diversity problem based on max-clique. *Computers & Operations Research* 36(8) 2429–2433.
- [6] Della Croce F., Garraffa M., Salassa F., 2014, A hybrid heuristic approach based on a quadratic knapsack formulation for the max-mean dispersion problem. *Lecture Notes in Computer Science*, pp. 186–197.
- [7] Duarte A., Martí R., 2007, Tabu search and grasp for the maximum diversity problem. *European Journal of Operational Research* 178(1), 71–84.
- [8] Duarte A., Sánchez-Oro J., Resende M.G.C., Glover F., Martí R., 2015, Greedy randomized search procedure with exterior path relinking for differential dispersion minimization. *Information Sciences* 296, 46–60.
- [9] Galinier P., Boujbel Z., Fernandes M. C., 2011, An efficient memetic algorithm for the graph partitioning problem. *Annals of Operations Research* 191(1), 1–22.
- [10] Glover F., Laguna. M., 1997, Tabu search. *Kluwer Academic Publishers*, Boston.
- [11] Glover F., Laguna, M., Martí, R., 2000, Fundamentals of scatter search and path relinking. *Control Cybernetics* 39, 653–684.
- [12] Glover F., Kuo C.C., Dhir K.S., 1998, Heuristic algorithms for the maximum diversity problem. *Journal of Information and Optimization Sciences* 19(1), 109–132.
- [13] Hao J.K., 2012, Memetic algorithms in discrete optimization. In F. Neri, C. Cotta, P. Moscato (Eds.) *Handbook of Memetic Algorithms. Studies in Computational Intelligence* 379, Springer, Chapter 6, pp. 73–94.
- [14] Kerchove C., Dooren P.V., 2008, The page trust algorithm: how to rank web pages when negative links are allowed? *Proceedings SIAM International Conference on Data Mining*, 346–352.
- [15] Martí R., Gallego M., Duarte A., 2010, A branch and bound algorithm for the maximum diversity problem. *European Journal of Operational Research* 200(1), 36–44.
- [16] Martí R., Sandoya F., 2013, GRASP and path relinking for the equitable dispersion problem. *Computers & Operations Research* 40(12), 3091–3099.

- [17] P. Moscato, C. Cotta. A gentle introduction to memetic algorithms. *In F. Glover and G. Kochenberger (Eds.), Handbook of Metaheuristics*, Kluwer, Norwell, Massachusetts, USA, 2003.
- [18] Neri F., Cotta C., Moscato P.(Eds.) Handbook of Memetic Algorithms. *Studies in Computational Intelligence 379*, Springer, 2011.
- [19] Porumbel D.C., Hao J.K., Glover F., 2011, A simple and effective algorithm for the MaxMin diversity problem. *Annals of Operations Research* 186(1), 275–293.
- [20] Prokopyev O.A., Kong N., Martinez-Torres D.L., 2009, The equitable dispersion problem. *European Journal of Operational Research* 197(1), 59–67.
- [21] Palubeckis G., 2007, Iterated tabu search for the maximum diversity problem. *Applied Mathematics and Computation* 189(1), 371–383.
- [22] Resende M.G.C., Martí R., Gallego M., Duarte A., GRASP and path relinking for the max–min diversity problem. *Computers & Operations Research* 37(3), 498–508
- [23] Saboonchi B., Hansen P., Perron S., 2014, MaxMinMin  $p$ -dispersion problem: A variable neighborhood search approach. *Computers & Operations Research* 52 (Part B), 251–259.
- [24] Wu Q.H., Hao J.K., 2013, A hybrid metaheuristic method for the maximum diversity problem. *European Journal of Operational Research* 231(2), 452–464.
- [25] Yang B., W. Cheung, Liu J., 2007, Community mining from signed social networks. *IEEE Transactions on Knowledge and Data Engineering* 19(10), 1333–1348.

Table 6

**Appendix:** Computational results of the proposed MAMMDP algorithm on the set of 100 small instances with  $n \leq 150$ . MAMMDP is run 20 times to solve each instance, each run being limited to 10 seconds. The previous best known results ( $f_{pre}$ ) are from: <http://www.optisicm.es/edp/>

Instance	n	$f_{pre}$	Memetic Algorithm			
			$f_{best}$	$f_{avg}$	SR	$t(s)$
MDP11_20	20	13.88	13.880000	13.880000	20/20	0.02
MDP12_20	20	13.608	13.608000	13.608000	20/20	0.02
MDP13_20	20	11.7957	11.795714	11.795714	20/20	0.02
MDP14_20	20	17.54	17.540000	17.540000	20/20	0.02
MDP15_20	20	16.0063	16.006250	16.006250	20/20	0.02
MDP16_20	20	14.6064	14.606364	14.606364	20/20	0.02
MDP17_20	20	14.8822	14.882222	14.882222	20/20	0.02
MDP18_20	20	14.4614	14.461429	14.461429	20/20	0.02
MDP19_20	20	14.035	14.035000	14.035000	20/20	0.02
MDPII0_20	20	13.4433	13.443333	13.443333	20/20	0.02
MDPII1_20	20	18.855	18.855000	18.855000	20/20	0.02
MDPII2_20	20	17.83	17.830000	17.830000	20/20	0.02
MDPII3_20	20	18.11	18.110000	18.110000	20/20	0.02
MDPII4_20	20	17.842	17.842000	17.842000	20/20	0.02
MDPII5_20	20	16.344	16.344000	16.344000	20/20	0.02
MDPII6_20	20	17.61	17.610000	17.610000	20/20	0.02
MDPII7_20	20	18.9383	18.938333	18.938333	20/20	0.02
MDPII8_20	20	21.88	21.880000	21.880000	20/20	0.02
MDPII9_20	20	19.785	19.785000	19.785000	20/20	0.02
MDPII10_20	20	22.599	22.599000	22.599000	20/20	0.02
MDP11_25	25	17.2708	17.270833	17.270833	20/20	0.03
MDP12_25	25	15.1214	15.121429	15.121429	20/20	0.02
MDP13_25	25	14.1817	14.181667	14.181667	20/20	0.03
MDP14_25	25	19.8567	19.856667	19.856667	20/20	0.03
MDP15_25	25	17.5371	17.537143	17.537143	20/20	0.02
MDP16_25	25	17.9667	17.966667	17.966667	20/20	0.03
MDP17_25	25	16.207	16.207000	16.207000	20/20	0.03
MDP18_25	25	18.1367	18.136667	18.136667	20/20	0.03
MDP19_25	25	17.4778	17.477778	17.477778	20/20	0.03
MDPII0_25	25	19.4592	19.459167	19.459167	20/20	0.03
MDPII1_25	25	21.81	21.810000	21.810000	20/20	0.03
MDPII2_25	25	22.185	22.185000	22.185000	20/20	0.03
MDPII3_25	25	23.5644	23.564444	23.564444	20/20	0.03
MDPII4_25	25	19.74	19.740000	19.740000	20/20	0.03
MDPII6_25	25	20.1744	20.174444	20.174444	20/20	0.03
MDPII7_25	25	19.947	19.947000	19.947000	20/20	0.03
MDPII8_25	25	23.921	23.921000	23.921000	20/20	0.03
MDPII9_25	25	25.016	25.016000	25.016000	20/20	0.03
MDPII10_25	25	23.575	23.575000	23.575000	20/20	0.03
MDP11_30	30	19.861	19.861250	19.861250	20/20	0.03
MDP12_30	30	18.813	18.813333	18.813333	20/20	0.03
MDP13_30	30	15.249	15.248889	15.248889	20/20	0.03
MDP14_30	30	22.717	22.717333	22.717333	20/20	0.03
MDP15_30	30	17.237	17.236667	17.236667	20/20	0.03
MDP16_30	30	18.375	18.375455	18.375455	20/20	0.03
MDP17_30	30	15.293	15.292500	15.292500	20/20	0.03
MDP18_30	30	19.247	19.247273	19.247273	20/20	0.04
MDP19_30	30	22.004	22.004286	22.004286	20/20	0.03
MDPII0_30	30	18.698	18.698462	18.698462	20/20	0.03
MDPII1_30	30	22.2721	22.272143	22.272143	20/20	0.03
MDPII2_30	30	26.9138	26.913846	26.913846	20/20	0.03
MDPII3_30	30	21.8973	21.897273	21.897273	20/20	0.04
MDPII4_30	30	20.5375	20.537500	20.537500	20/20	0.03
MDPII5_30	30	22.79	22.790000	22.790000	20/20	0.03
MDPII6_30	30	20.351	20.351000	20.351000	20/20	0.03
MDPII7_30	30	27.655	27.655000	27.655000	20/20	0.03
MDPII8_30	30	26.8842	26.884167	26.884167	20/20	0.03
MDPII9_30	30	24.1767	24.176667	24.176667	20/20	0.03
MDPII10_30	30	24.8	24.800000	24.800000	20/20	0.03
MDP11_35	35	19.1833	19.183333	19.183333	20/20	0.04
MDP12_35	35	17.168	17.168000	17.168000	20/20	0.04
MDP13_35	35	17.0746	17.074615	17.074615	20/20	0.04
MDP14_35	35	23.35	23.350000	23.350000	20/20	0.04
MDP15_35	35	19.0177	19.017692	19.017692	20/20	0.04
MDP16_35	35	19.445	19.445000	19.445000	20/20	0.04
MDP17_35	35	19.4971	19.497143	19.497143	20/20	0.04
MDP18_35	35	21.2307	21.230667	21.230667	20/20	0.04
MDP19_35	35	20.98	20.980000	20.980000	20/20	0.04
MDPII0_35	35	16.9378	16.937778	16.937778	20/20	0.04
MDPII1_35	35	25.968	25.967500	25.967500	20/20	0.04
MDPII2_35	35	26.135	26.135455	26.135455	20/20	0.04
MDPII3_35	35	24.159	24.159231	24.159231	20/20	0.04
MDPII4_35	35	24.415	24.415000	24.415000	20/20	0.04
MDPII5_35	35	23.857	23.857500	23.857500	20/20	0.04
MDPII6_35	35	24.673	24.673077	24.673077	20/20	0.04
MDPII7_35	35	29.394	29.393846	29.393846	20/20	0.04
MDPII8_35	35	25.297	25.217273	25.217273	20/20	0.04
MDPII9_35	35	27.435	27.435000	27.435000	20/20	0.04
MDPII10_35	35	25.712	25.712500	25.712500	20/20	0.04
MDP11_150	150	45.92	45.920192	45.920192	20/20	0.17
MDP12_150	150	43.39	43.392381	43.392381	20/20	0.19
MDP13_150	150	40.05	40.046304	40.046304	20/20	0.17
MDP14_150	150	44.04	44.044138	44.044138	20/20	0.17
MDP15_150	150	42.48	42.479388	42.479388	20/20	0.17
MDP16_150	150	43.72	43.722955	43.722955	20/20	0.17
MDP17_150	150	46.08	46.077308	46.077308	20/20	0.16
MDP18_150	150	42.45	42.451346	42.451346	20/20	0.18
MDP19_150	150	42.48	42.479767	42.479767	20/20	0.18
MDPII0_150	150	41.8	41.797805	41.797805	20/20	0.16
MDPII1_150	150	57.48	57.484000	57.484000	20/20	0.16
MDPII2_150	150	57.82	57.820652	57.820652	20/20	0.17
MDPII3_150	150	58.42	58.421818	58.421818	20/20	0.17
MDPII4_150	150	57.38	57.381064	57.381064	20/20	0.17
MDPII5_150	150	54.23	54.228571	54.228571	20/20	0.17
MDPII6_150	150	56.44	56.442653	56.442653	20/20	0.16
MDPII7_150	150	58.89	58.889167	58.889167	20/20	0.17